Noname manuscript No.
(will be inserted by the editor)

An Agent for Learning New Natural Language Commands

Amos Azaria · Shashank Srivastava · Jayant Krishnamurthy · Igor Labutov · Tom M. Mitchell

the date of receipt and acceptance should be inserted later

Abstract Teaching via natural language is an intuitive way for end users to add functionality to a virtual assistant, enabling them to personalize their assistant with new commands without requiring the intervention of the system developer, who cannot possibly anticipate all of an end user's needs. In this paper we introduce our Learning by Instruction Agent (LIA), the first virtual assistant, for an email domain, that is capable of learning how to perform new commands taught by end users in natural language. LIA grounds the semantics of each command in terms of primitive executable procedures. When a user provides LIA with a command that it does not understand, it prompts the user to explain the command through a sequence of natural language steps. From this input, LIA learns the meaning of the new command and how to generalize the command to novel situations. For example, having been taught how to "forward an email to Alice", it can correctly understand "forward this email to Bob". We show that users that were assigned to interact with LIA completed the task quicker than users assigned to interact with a non-learning agent. These results demonstrate the potential of natural language teaching to improve the capabilities of intelligent personal assistants.

Amos Azaria
Department of Computer Science, Ariel University, Israel
Data Science Center, Ariel University, Israel
E-mail: amos.azaria@ariel.ac.il

Shashank Srivastava
Microsoft Research, Redmond, WA

Jayant Krishnamurthy
Semantic Machines, Berkeley, CA

Igor Labutov
LAER AI, New York, NY

Tom M. Mitchell
Machine Learning Department, Carnegie Mellon University, Pittsburgh, PA

We annotated 4759 natural language statements with their associated computer readable execution commands (logical forms) to form a dataset (which we publicize in this paper). We present the performance of several different parser methods on this dataset.

1 Introduction

Major companies are exploring intelligent agents such as Siri, Google Assistant, Cortana and Echo, and we expect that they will continue to further develop them. These intelligent agents have demonstrated great potential for assisting users with basic daily tasks. However, unlike a human assistant, interactions with these agents are usually limited to commands pre-programmed by the developers. While some assistant platforms have an open API that allow third party developers to add "skills" to the assistant, these skills are pre-programmed, and can only be developed by programmers, not by the layman, and obviously not by means of natural language. A teachable agent has two significant benefits over such pre-programmed agents. First, teaching enables a user to define a new command and a sequence of actions to perform it, including commands that were not considered by the software developer. Second, teaching can be used to expand the range of natural language phrases understood by the system (e.g., by teaching the system that "drop a note to Bill" has the same meaning as "send an email to Bill".). Together, these two capabilities might enable a community of users to jointly instruct a system to learn a significant number of new commands, and ways to express them in natural language.

When studying the past of computers, starting from the previous century, we can observe that in the 40's computers could merely serve as calculators, that is, computers could perform a (very limited) set of tasks, which could not be extended in any way (without rebuilding the computer). In the 50's, computers could be programmed using assembly, which was a great advancement, as computer functionality could be extended and computers could perform many new tasks. In the 60's programming languages were developed which allowed a broader range of people to program a computer and further extend its capabilities. In the 70's software libraries became popular, which allowed programmers to build upon software written by others, further enlarging the population of programmers and further extending computers' capabilities. The rise of the internet in the late 90's allowed easier access to these libraries, as well as access to knowledge and information that has allowed more people to write programs and macros more easily.

The last decade can be seen as the rise of machine learning, a great leap for artificial intelligence. Machine learning provides numerous applications in many different fields such as genome sequencing and classification, medical diagnostics, fraud detection, recommender systems, search engines, information and opinion retrieval, autonomous driving, personal assistants and many more. We believe that an important advancement in artificial intelligence will be instructability, that is, an instructable agent that can learn from *any* user to perform any task, or perceive a concept, using natural language dialog.

This paper presents our Learning by Instruction Agent (LIA), an intelligent agent for the email domain that allows users to teach it new commands solely using natural language interactions. LIA operates in an email environment, where it learns to interpret natural language commands in terms of given primitive actions such as sending emails. Users teach LIA to perform a new command by providing step-by-step instructions. LIA is then able to generalize and later execute this command with different parameters; for example, having been taught "forward an email to Alice," LIA can correctly interpret "forward this email to Bob." Users can also teach LIA declarative knowledge by defining new concepts and relations between them, e.g., defining the notion of a contact, and that each contact has an associated email address.

We evaluated the performance of LIA, by running a user study with 131 users. We randomly assigned every user to one of two group. The first group allowed the user to interact with LIA (the learning agent), while the second group interacted with a degenerated version of the agent that did not allow the users to teach it new commands. The users interacting with LIA taught it new commands, which they later reused. We show that users that were assigned to interact with the learning agent, completed the task quicker than users assigned to interact with a non-learning agent. The users interacting with LIA also reported the interaction to be more interesting than those interacting with the non-learning agent. We annotated 4759 commands given to LIA with machine readable commands (logical-forms) and evaluated the performance of several parsers on this dataset including a deep learning based parser.

The contributions of this work are:

1. A working implementation of an agent that can be taught via natural language to perform new commands.
2. A novel lexicon induction algorithm that enables the agent to generalize from a taught command to unseen commands.
3. A controlled experiment showing that users interacting with LIA completed the task quicker than those interacting with a non-learning agent.
4. An annotated dataset of conversations, comprising 4759 natural language statements with their associated computer readable execution commands (logical forms), available at `http://azariaa.com/content/LabeledDialogData.zip`. We use this dataset to create a context-based deep learning parser.

2 Background

Much evidence exists that human interaction may assist machine learning [25,30,50]. The most common and widely spread method for humans to assist machine learning is by labeling existing data. This labeled data is then used by machine learning methods to classify new data. Unfortunately, methods that further exploit human knowledge are much less common. Most learning methods that exist today, which exploit human knowledge or feedback, fall into one of the following:

- A method that was common in the 80's includes elicitation of "if-then" rules from human experts to create expert systems [23].
- Another method includes eliciting probability estimations from humans along with dependencies between variables, i.e., how one variable (e.g. genotype) may depend on another (e.g. symptom), or to what extent it may be influenced by it. This information is usually used to help construct a probabilistic graphical model which then allows inference about new utterances [32]. Experts may also be used by providing information on how state-action pairs are assumed to be similar in an MDP settings [49], or in which states different actions might lead to very different results [55].
- The third method which is less common includes humans providing reinforcement feedback to an agent. One example is a human teaching an agent how to play Tetris by giving it positive and negative feedback every time it performs an action which the human believes to be good or bad, respectively [30].
- Behavioral cloning [45] is a method that allows the creation of an agent that imitates human behavior. The common use of behavioral cloning is to apply supervised learning in

order to learn a policy that imitates human behavior. This policy may be used as a seed for a reinforcement learner. Similarly, Learning from observations (LfO) [52], are defined as learning from a sporadic collection of natural interactions with the environment, i.e. input by humans for their specific goals not aiming at teaching the agent.
- Inverse reinforcement learning [1] allows an agent to extract a reward function from examples of how a human performs a task (or plays a game).
- Programming by demonstration (PbD) is most commonly used in the human-robot interaction community and is described below.

Of the above, programming by demonstration is probably the most closely related to our work. Much of this research derives from the human-robot interaction (HRI) community (although some works do not involve robots [42,2]). The common case study in PbD is with a person trying to show a robot how to lift or select a certain object, move an object or perform some other task [4]. In most studies the human teacher actually moves the robot's arms to perform the taught task [17], or controls the robot using a control peg, while in some other studies, the human teacher performs the task in front of the robot's cameras [43], or wears data gloves [35]. In many cases the robot can also generalize beyond the specific training scenario, to perform the task also in different conditions. For example, Calinon et al. [17], teach a robot by demonstrating how to move a chess piece on a board of chess by moving the robot's arms. The robot can then generalize and move the same piece also when it is located in a different location. In previous work we introduced an agent that learns how to execute natural language commands on a mobile phone by demonstration [39]. Billard et al. [13] state that learning by demonstration has three advantages: First, PbD reduces the search space for the robot, allowing the robot to find a way to perform the task; Secondly, PbD is more natural and easier for the human teacher, in comparison to tedious software programming; Thirdly, PbD couples the perception (or vision) and action (or robot motors). Similarly to programming by demonstration, in our work, the user teaches the agent how to complete a new command, but instead of demonstrating this new command, the user explains step by step, in natural language, how to complete the new command.

In this work, we present an agent that can learn from a single example, while programming by demonstration usually requires several training examples. A notable exception is the work by Allen et al. [2] who built a system with a complex graphical interface that poses questions to the teacher as it learns a new task (in the information retrieval domain). Their system succeeded in learning 30 out of 55 tasks based on a single demonstration by a human user for each task. However, as they state, users did not find the system easy to use, and each user required a full work day to teach approximately 3 tasks.

The work of She et al. [54], seems to be very close to the work presented in this paper, as they use natural language as well to teach a robot a new task. She et al. developed a system that allows a human user to teach a robot to perform basic actions in the world of blocks [60] using a natural language dialog. Initially, the robot can only execute 3 commands: Open_Grip, Close_Grip and MoveTo. Using these basic commands, users were required to teach the robot the following more complex commands: Pickup, Grab, Drop, ClearTop and Stack.

There have been several works on programming by demonstration in systems using graphical user interfaces (rather than robots). Koedinger et al. [31] built a system with a graphical interface that allowed an author to create arithmetic problems and demonstrate their solutions as well as common errors. The intelligent tutor created a solution graph and allowed the author to annotate this graph with hints and feedback. The intelligent tutor used these examples to tutor a student providing hints and feedback.

Learning new commands requires the agent to understand the meaning of these commands, therefore, another field closely related to our work is Grounded language acquisition [28, 20]. In this field the main challenges are extracting the semantic meaning of words and sentences and connecting language to action and perception. Unlike PbD, in this field, the system usually learns without having a human in the loop. One example is the work by Chen and Mooney [20], in which a system learns how to sportscast a RoboCup simulated game and outputs statements, for example, that a certain player has a ball, that the ball was passed on to a different player and that a player has intercepted this pass. The system learns from data which includes many games and the text description of what is shown in the game, not having any additional domain or language background. The system builds a generative model based on probabilistic context-free grammar (PCFG), which maximizes the likelihood of the data, and uses this model to sportscast a new, unseen, game. Chen and Mooney, also apply a similar method to the domain of navigation [21]. In this domain, a user must navigate via a 3-D maze with objects located in it and reach a specific goal. The data in this domain is composed of a set of paragraphs in natural language, each associated with a video which shows the execution of this paragraph. This learning method may be categorized as learning by observation. Branavan et al. [14] introduce a reinforcement learning based method for grounded language acquisition. Their proposed agent interacts in the domain of troubleshooting and tries to map documents to a set of actions represented by these documents. The agent requires the documents to be annotated so that the agent receives a reward on its execution.

Several works focus on allowing an agent to understand the meaning of a given command based on the context or provided labels (without explicitly being taught how to execute this command by the user). Thomason et al. [58] use CCG parsing on natural language commands in order for a robot to execute them. They use conversations with previous users to better understand user commands as well as overcome typos and spelling mistakes. Quirk et al. [47] converted “recipes” written in natural language to if-then statements, in which the “if” part is a condition on a sensor (e.g. phone camera, mic. etc) or a cyber-sensor (e.g. weather, Twitter etc.), and the “then” part is bounded to a command (e.g. opening the camera app, or making a sound). There has also been some preliminary work on translating functionality descriptions in natural language into scripts by Le et al. [38], and translation of conditions into if-then clauses [46].

Cantrell et al. [18] design an architecture for a robot that enables it to interact with a user using a natural language dialog. The robot learns by instruction preconditions and post-conditions of a command taught by the user. For example, the user can teach the robot that pushing on a door changes the door’s state from closed to opened. This knowledge is later used by the robot’s motion planner module to plan a set of actions. This is different from this work in which the agent is taught how to execute the command and not its preconditions and post-conditions.

Supervised semantic parsing has been studied in a wide range of settings [62, 61, 37]. Recent approaches have focused on various strategies for using weaker forms of supervision [24, 34, 11] and rapid prototyping of semantic parsers for new domains [59, 44]. Other works have explored semantic parsing in grounded contexts, and using perceptual context to assist semantic parsing [41, 33]. In terms of the approach, while our formulation is largely agnostic to the choice of semantic parsing framework, for this work our method is based on CCG semantic parsing, which is a popular semantic parsing approach [63, 36, 5, 58]. The CCG grammar formalism [57] has been widely used for its explicit pairing of syntax with semantics, and allows expression of long range dependencies extending beyond context-free-grammars.

Another field that is related to this research is natural language programming [12], in which a programmer can use natural language to develop software. In Inform 7 [48], for example, a programmer can create an interactive fiction program using statements which are actually valid English sentences. Some examples of such statements in the Inform 7 language are "The kitchen is a room," "There is a stove in the kitchen," and "The description of the stove is "very dirty" ". These statements are required to be in a very specific form so that later they can be compiled into an interactive fiction. However, while Inform 7 allows the usage of predefined natural language statements to create an interactive fiction, it does not provide any method for teaching its compiler how to compile new statements (neither using a graphical interface nor using natural language).

Our work is also closely related to the field of human-agent interaction [16,6] (more specifically, works that include a natural language interface [51,9]), human modeling [19] and agents for teaching humans [3,53]. LIA, the agent presented in this paper, was first introduced in [8] and was extended to a mobile phone app, which receives its commands by speech, in [22].

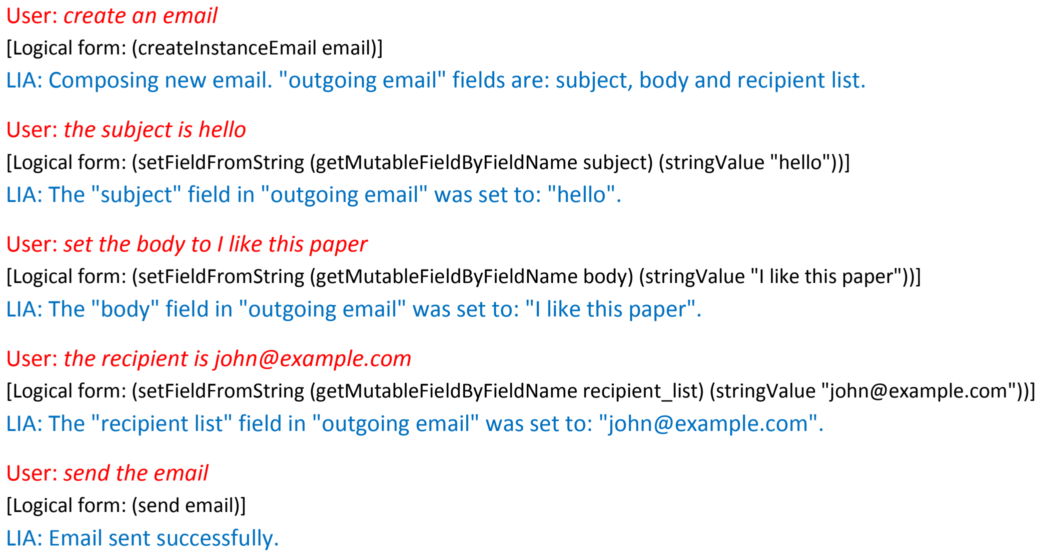
3 The Agent (LIA)

LIA, our instructable agent operates in an email domain, where the basic actions include reading, composing, and sending emails. We chose this domain as our research environment because email manipulation is a common use of mobile phones where we anticipate that an instructable agent may be useful. The user interacts with LIA in a text dialogue by giving it commands in natural language, LIA responds in natural language and takes various actions to fulfill the user's command. LIA is built from two components that enable it to intelligently respond to user commands: a semantic parser which assigns executable semantics to each natural language command, and a back-end which executes these commands. The back-end contains a number of built-in, executable functions understood by LIA, such as `sendEmail`, along with a declarative knowledge base containing statements in predicate logic. LIA interprets commands using a semantic parser that maps each command to a *logical form* – a program in a Lisp-like language – containing one or more of these functions and predicates. This logical form represents the semantics of the user command, and is evaluated (executed) by the back-end to produce a response. See Figure 1 for an example of an interaction in which the user composes an email and sends it.

LIA can be instructed in two distinct fashions, corresponding to two distinct types of knowledge that LIA can acquire. First, the user can teach LIA new *declarative knowledge* by defining new concepts, along with fields and instances of those concepts. For example, the user can define the concept "contact" and further state that "a contact has an email address" and "bob is a contact." LIA processes these interactions by adding new concepts, fields and instances to its knowledge base (as performed in [26]). See Figure 2 for an example of an interaction in which the user teaches LIA about a contact and an email address.

Second, using our novel approach, the user can teach LIA new *procedural knowledge*, i.e., how to execute a new command. For example, the user can teach the system how to "forward" an email by providing natural language instructions that map to actions the system already understands. LIA learns new procedural knowledge from these interactions using a novel lexicon induction algorithm that updates the semantic parser. The updated parser is then able to understand both the taught command as well as the unseen, but similar commands. See Figure 3 for an example of an interaction in which the user teaches LIA how to reply to an email.

Fig. 1 An example of an interaction in which a user composes an email and sends it. The logical form into which each user command was parsed, appears inside the brackets.



User: *create an email*
[Logical form: (createInstanceEmail email)]
LIA: Composing new email. "outgoing email" fields are: subject, body and recipient list.

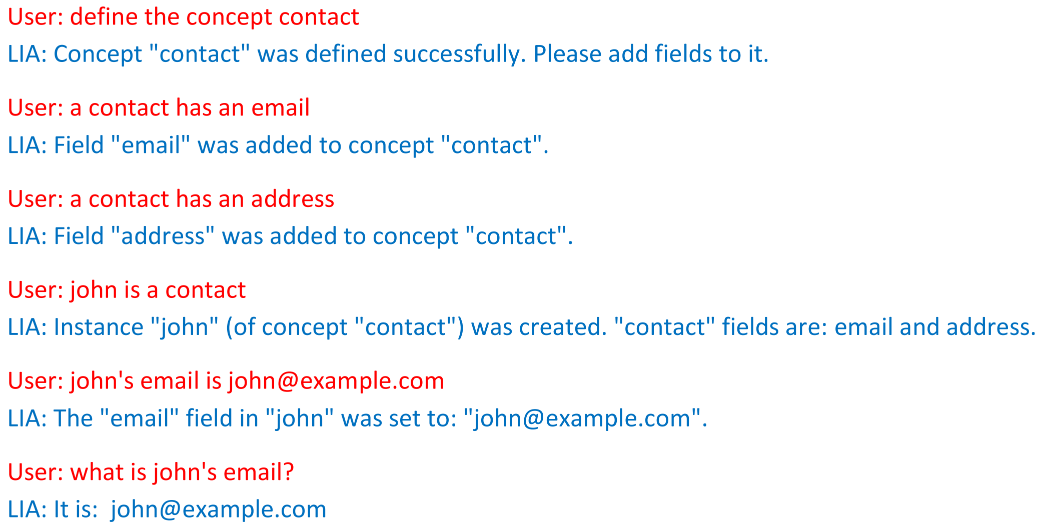
User: *the subject is hello*
[Logical form: (setFieldFromString (getMutableFieldByFieldName subject) (stringValue "hello"))]
LIA: The "subject" field in "outgoing email" was set to: "hello".

User: *set the body to I like this paper*
[Logical form: (setFieldFromString (getMutableFieldByFieldName body) (stringValue "I like this paper"))]
LIA: The "body" field in "outgoing email" was set to: "I like this paper".

User: *the recipient is john@example.com*
[Logical form: (setFieldFromString (getMutableFieldByFieldName recipient_list) (stringValue "john@example.com"))]
LIA: The "recipient list" field in "outgoing email" was set to: "john@example.com".

User: *send the email*
[Logical form: (send email)]
LIA: Email sent successfully.

Fig. 2 An example of an interaction in which a user teaches LIA about a contact and an email address, and defines a new contact.



User: define the concept contact
LIA: Concept "contact" was defined successfully. Please add fields to it.

User: a contact has an email
LIA: Field "email" was added to concept "contact".

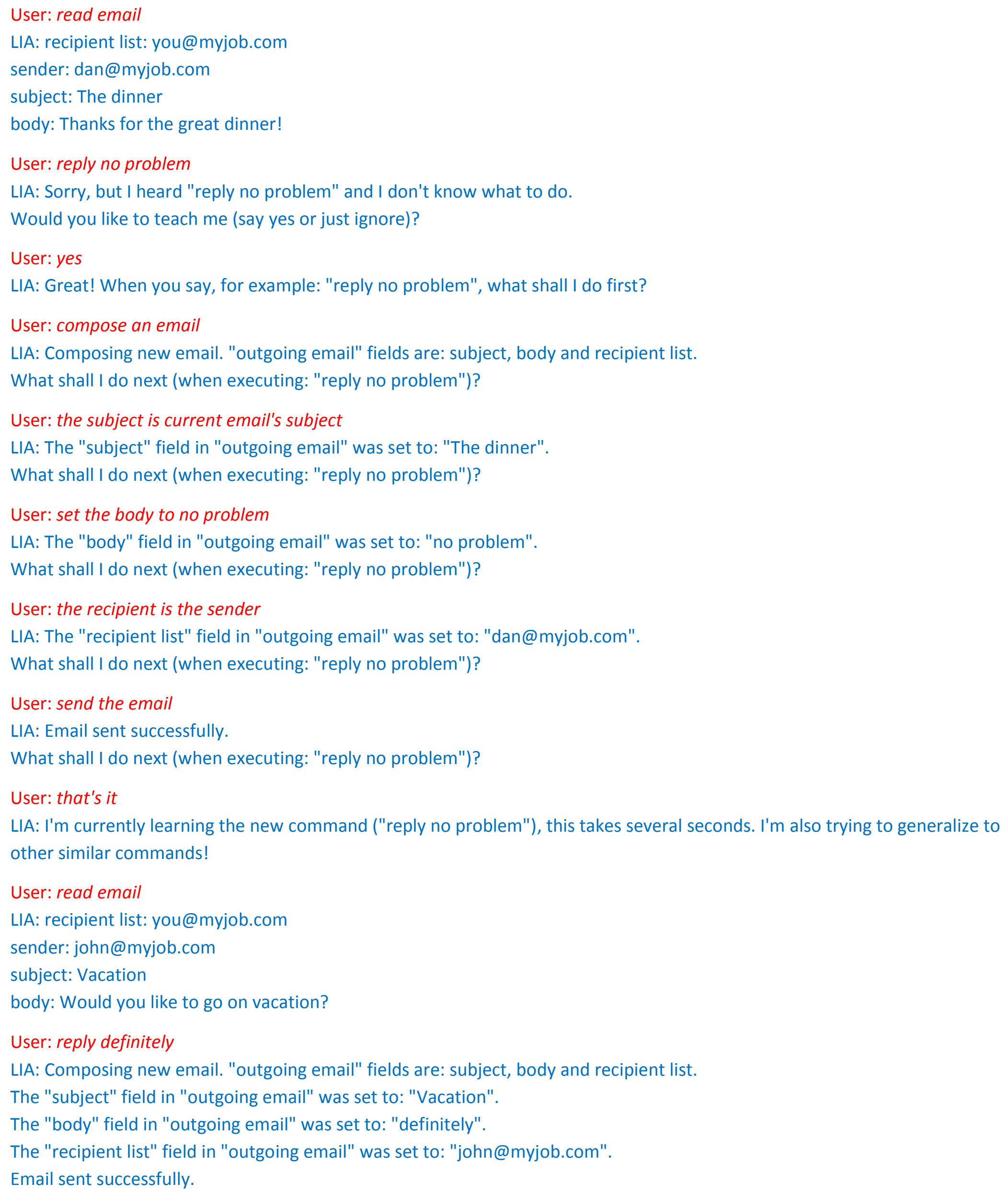
User: a contact has an address
LIA: Field "address" was added to concept "contact".

User: john is a contact
LIA: Instance "john" (of concept "contact") was created. "contact" fields are: email and address.

User: john's email is john@example.com
LIA: The "email" field in "john" was set to: "john@example.com".

User: what is john's email?
LIA: It is: john@example.com

Fig. 3 An example interaction in which a user teaches LIA how to reply to an email, and uses this new command with different parameters.



User: *read email*
LIA: recipient list: you@myjob.com
sender: dan@myjob.com
subject: The dinner
body: Thanks for the great dinner!

User: *reply no problem*
LIA: Sorry, but I heard "reply no problem" and I don't know what to do.
Would you like to teach me (say yes or just ignore)?

User: *yes*
LIA: Great! When you say, for example: "reply no problem", what shall I do first?

User: *compose an email*
LIA: Composing new email. "outgoing email" fields are: subject, body and recipient list.
What shall I do next (when executing: "reply no problem")?

User: *the subject is current email's subject*
LIA: The "subject" field in "outgoing email" was set to: "The dinner".
What shall I do next (when executing: "reply no problem")?

User: *set the body to no problem*
LIA: The "body" field in "outgoing email" was set to: "no problem".
What shall I do next (when executing: "reply no problem")?

User: *the recipient is the sender*
LIA: The "recipient list" field in "outgoing email" was set to: "dan@myjob.com".
What shall I do next (when executing: "reply no problem")?

User: *send the email*
LIA: Email sent successfully.
What shall I do next (when executing: "reply no problem")?

User: *that's it*
LIA: I'm currently learning the new command ("reply no problem"), this takes several seconds. I'm also trying to generalize to other similar commands!

User: *read email*
LIA: recipient list: you@myjob.com
sender: john@myjob.com
subject: Vacation
body: Would you like to go on vacation?

User: *reply definitely*
LIA: Composing new email. "outgoing email" fields are: subject, body and recipient list.
The "subject" field in "outgoing email" was set to: "Vacation".
The "body" field in "outgoing email" was set to: "definitely".
The "recipient list" field in "outgoing email" was set to: "john@myjob.com".
Email sent successfully.

3.1 Back-end Command Executor

LIA has a back-end that can evaluate logical forms (lambda expressions) incorporating 45 primitive, executable functions. Some examples are sendEmail, which sends the composed email, setFieldFromFieldVal, which sets a field from an evaluation of a different field, addFieldToConcept which adds a field to a concept, createInstanceByConceptName which creates an instance and deleteInstance which deletes an instance. Two additional notable functions that will be discussed later are unknownCommand and teachNewCommand, both of which start a dialog that lets the user teach a new command.

Upon execution, the back-end also builds a user-friendly response that either indicates which action was just performed (e.g., "The subject field of the outgoing email was set to party time for all"), or, in case of failure provides an informative statement conveying the reason for failure and sometimes suggesting a possible correction. For example, if the user says "set momthebest7@bestforyou.com," LIA replies "Sorry, but I don't know what should be set to momthebest7@bestforyou.com. Please repeat and tell me what should be set to it (e.g., set example to momthebest7@bestforyou.com)."

3.2 Semantic Parser

LIA uses a Combinatory Categorial Grammar (CCG) semantic parser to map natural language commands to logical forms containing functions and concepts executable by the back-end. CCG is often used to build semantic parsers due to its tight coupling of syntax and semantics [62]. CCG grammars are more expressive than context-free grammars, and are able to represent long-range dependencies present in some linguistic constructions, such as relative clauses, that cannot be represented in context-free formalisms [57].

A CCG semantic parser has three parts: a lexicon, a set of grammar rules, and a trained parameter vector. The lexicon is a table mapping words to syntactic categories and logical forms (see Table 2). The intuition of CCG is that, syntactically and semantically, *words behave like functions*. Thus, syntactic categories represent function type specifications, where the argument type appears on the right of the slash and the return type on the left. The direction of the slash determines on which side of the syntactic category each argument must appear. For example, the syntactic category ((S\PP_StringV)/MutableField) accepts a MutableField on the right, followed by a PP_StringV on the left, and returns an S. In this fashion, the syntactic category of a lexicon entry specifies how it can combine with other words during parsing. Our CCG parser also permits strings from the command to enter the parse with the syntactic category StringN or StringV, and words in the sentence to be skipped. This second capability is used to ignore function words in the sentence that contribute little to the overall meaning.

Parsing in CCG derives syntactic categories and logical forms for phrases from their constituent parts by applying a small number of grammar rules. These rules correspond to standard function operations, such as application and composition. For example, the logical forms for "send" and "email" in Table 2, "(lambda x (send x))" and "email" respectively, can be combined using function application to derive the second example in Table 1, "(send email)". Our grammar also includes a small number of unary rules that represent common implicit conversions between types. See Table 3 for examples of unary rules.

Together, the lexicon and the grammar rules define a set of possible parses for every input command, each of which may have a different logical form. In order to select the single best parse, the semantic parser is trained using a data set of commands paired with their corresponding logical forms (see Table 1). First, we define a feature function ϕ that maps a CCG parse t of a command s to a feature vector $\phi(t, s)$. Our features include indicator features for the lexicon entries used in the parse, the parse's function/argument applications, and various features derived from the string itself, such as the part-of-speech tags (obtained using CoreNLP [40]), the number of words, etc. During training, the parser learns a parameter vector θ that assigns a high score $\theta^T \phi(t, s)$ to correct parses. At test time, the parser selects the highest-scoring parse for each command, i.e., the parse, t, that maximizes $\theta^T \phi(t, s)$. For more information about CCG semantic parsing, including details of parsing and training algorithms, we refer the reader to Zettlemoyer and Collins [62].

LIA's semantic parser has over 300 lexicon entries, 14 unary rules, and was trained using 150 training examples. These lexicon entries were manually defined in order to support fluent communication with a human user in the email domain. Defining lexicon entries, unary rules and providing training examples requires: understanding the way a CCG parser works; familiarity with functions supported by LIA, used in the back-end to evaluate the resulting logical form; and the understanding of the commands users might want to use. LIA learns new lexicon entries (and adds them to its lexicon entry list) using our lexicon induction algorithm described in Section 3.4. While learning new commands, LIA also enlarges its training examples.

Table 1 Training examples for the semantic parser that consists of a natural language command paired with a logical form, which is a computer program written in a Lisp-like language composed of one or more basic actions that LIA can perform. The top four examples are part of the default training set and the bottommost example was generated by a user teaching interaction.

Text Command	Logical Form
set the subject to time to go	(setFieldFromString (getMutableFieldByFieldName subject) (stringValue "time to go"))
send the email	(send email)
set body to email's body and send email	(doSeq (setFieldFromFieldVal (getMutableFieldByFieldName body) (evalField (getFieldByInstanceNameAndFieldName email body))) (send email))
add length as a field in table	(addFieldToConcept table (stringNoun "length"))
forward to charlie	(doSeq (doSeq (doSeq (doSeq (createInstanceByConceptName outgoingemail) (setFieldFromFieldVal (getMutableFieldByFieldName subject) (evalField (getFieldByInstanceNameAndFieldName email subject)))) (setFieldFromFieldVal (getMutableFieldByFieldName body) (evalField (getFieldByInstanceNameAndFieldName email body)))) (setFieldFromFieldVal (getMutableFieldByFieldName recipient) (evalField (getFieldByInstanceNameAndFieldName charlie email)))) (sendEmail))

3.3 Logical Form Evaluation

The logical forms output by the semantic parser are evaluated by the back-end in a standard Lisp fashion. Each argument of a function application is recursively evaluated in left-to-right order. For example, to interpret the first example command in Table 1, LIA first evaluates

Table 2 Lexicon entries required to parse the examples in Table 1. The syntactic category specifies how each word can combine with adjacent words and phrases during parsing and the logical form gives the word's meaning. The bottommost entry was automatically produced by lexicon induction when LIA was taught how to "forward to charlie."

Word	Syntactic Category	Logical Form
set	((S/PP_StringV)/MutableField)	(lambda x y (setFieldFromString x y))
to	PP_StringV/StringV	(lambda x x)
subject	FieldName	subject
send	S/InstanceName	(lambda x (send x))
email	InstanceName	email
set	((S/PP_FieldVal)/MutableField)	(lambda x y (setFieldFromFieldVal x y))
to	PP_FieldVal/FieldVal	(lambda x x)
and	(S/S)\S	(lambda x y (doSeq x y))
's	((Field\InstanceName)/FieldName)	(lambda x y (getFieldByInstanceNameAndFieldName y x))
forward	(S/InstanceName)	(lambda x (doSeq (doSeq (doSeq (doSeq (createInstanceByConceptName outgoingemail) (setFieldFromFieldVal (getMutableFieldByFieldName subject) (evalField (getFieldByInstanceNameAndFieldName email subject)))) (setFieldFromFieldVal (getMutableFieldByFieldName body) (evalField (getFieldByInstanceNameAndFieldName email body)))) (setFieldFromFieldVal (getMutableFieldByFieldName recipient) (evalField (getFieldByInstanceNameAndFieldName x email)))) (sendEmail)))

Syntactic Input	Syntactic Output	Semantics
FieldName	MutableField	(lambda x (getMutableFieldByFieldName x))
Field	FieldVal	(lambda x (evalField x))

Table 3 Examples of unary rules (required to parse the examples in Table 1).

getMutableFieldByFieldName and subject, then performs a function application with the corresponding values. The result of this application is the subject of the outgoing email. Next, LIA evaluates the expression containing stringValue, which returns the string "time to go." Finally, LIA calls setFieldFromString with the outgoing email's subject and the string "time to go." The result of evaluation is to set the outgoing email's subject to "time to go." LIA supports 45 primitive functions.

3.4 Learning New Commands

LIA learns new commands through an instruction dialogue that is initiated whenever the user enters a command that LIA does not understand, see Figure 3 for an example. In this case, the semantic parser outputs the logical form (unknownCommand). One of the entries in the unary rules allows the parser to evaluate *any* sentence as an (unknownCommand). However, the parser learns (from the training examples) that this rule should not be used too often. In the training examples some commands are provided as such that should be parsed into an unknown command; these commands are intentionally different than any command we assumed a user would teach LIA.

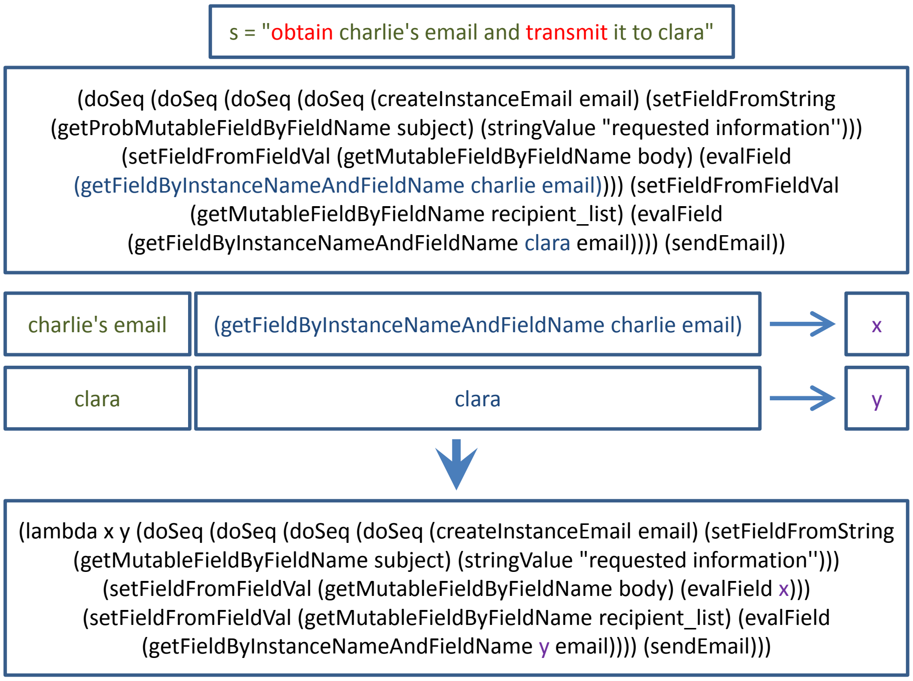
Evaluating the logical form (unknownCommand) starts an instruction interaction that enables the user to teach LIA how to execute the given command. LIA asks the user what it should do first in order to execute the command being learned, and then asks for consecutive actions, one at a time. The user responds with a sequence of natural language commands that LIA should perform. LIA parses and evaluates each command in the sequence during instruction to confirm that these commands can be performed. If, during this learning phase, LIA does not understand any of these commands (i.e. a command is evaluated as an unknown command), LIA replies that it does not understand that command and asks for an alternative command to execute and associate with the command being learned. Once the user ends the instruction interaction, LIA has a sequence of logical forms that, when evaluated sequentially, produces the desired result for the given command. LIA combines this sequence into a single logical form using the function doSeq (see Table 1, bottom). While executing a logical form with the doSeq function, LIA first evaluates and executes the first argument of the doSeq function, and then evaluates and executes its second argument. If execution fails on any first argument (for example, by setting an email recipient to a string that is not formatted as an email), execution is terminated and the second argument is not executed. LIA uses multiple doSeq calls in order to concatenate multiple commands together.

This alone does not allow the agent to generalize beyond this training command. Since a part of the new command given may be an argument (such as the words "no problem" in the dialog that appears in Figure 3, being an argument of the "reply" command being taught). LIA must understand what part of the command is actually an argument. When the user later uses this command, however, with a different argument, in our example the user says "reply definitely", LIA must know that it should execute the "reply" command but use the word "definitely" as the argument (instead of "no problem"). To that end LIA uses a novel *lexicon induction* algorithm to update the semantic parser to generalize the instruction to interpret other, similar commands. This algorithm learns which words in the taught command correspond to each part of the complete logical form. It first parses the taught command with the current semantic parser and examines the 100 best parses of each span of the command. If a span's logical form is a subexpression of the complete logical form, then it could be an argument that should be filled during parsing. For example, in the command "forward to charlie," the text span "charlie" parses to the logical form charlie, which is a subexpression of the complete logical form (Table 1, bottom). The algorithm finds possible arguments and removes them from the complete logical form to construct a set of candidate logical forms, then creates lexicon entries by pairing each of these candidates with every non-stopword in the command. The list of stopwords was obtained from CoreNLP[40].

In our example, the algorithm creates the final entry in Table 2, where charlie has been extracted as an argument to "forward". These lexicon entries are added to the lexicon, the command/logical form pair is added to the training set, and the parser is re-trained. Pseudocode for lexicon induction is provided as Algorithm 1.

We now provide a walk-though on the algorithm with a slightly more complex example. Suppose the given command was s ="obtain charlie's email and transmit it to clara". Suppose that the sequence of commands given by the user is translated to the following logical form: "(doSeq (doSeq (doSeq (doSeq (createInstanceEmail email) (setFieldFromString (getProbMutableFieldByFieldName subject) (stringValue "requested information"))) (setFieldFromFieldVal (getMutableFieldByFieldName body) (evalField (getFieldByInstanceNameAndFieldName charlie email)))) (setFieldFromFieldVal (getMutableFieldByFieldName recipient_list) (evalField (getFieldByInstanceNameAndFieldName clara email)))) (sendEmail))". When executing this logical form, the agent will send an email to

Fig. 4 Lexicon induction algorithm executed on the command “obtain charlie’s email and transmit it to clara”. The algorithm recognizes the expressions “charlie’s email” and “clara” as parameters, and substitutes their associated logical forms with parameters (x and y). The words “obtain” and “transmit” are recognized as activation words.



```
s = "obtain charlie's email and transmit it to clara"
```

```
(doSeq (doSeq (doSeq (doSeq (createInstanceEmail email) (setFieldFromString
(getProbMutableFieldByFieldName subject) (stringValue "requested information")))
(setFieldFromFieldVal (getMutableFieldByFieldName body) (evalField
(getFieldByInstanceNameAndFieldName charlie email)))) (setFieldFromFieldVal
(getMutableFieldByFieldName recipient_list) (evalField
(getFieldByInstanceNameAndFieldName clara email)))) (sendEmail))
```

```
(lambda x y (doSeq (doSeq (doSeq (doSeq (createInstanceEmail email) (setFieldFromString
(getMutableFieldByFieldName subject) (stringValue "requested information")))
(setFieldFromFieldVal (getMutableFieldByFieldName body) (evalField x)))
(setFieldFromFieldVal (getMutableFieldByFieldName recipient_list) (evalField
(getFieldByInstanceNameAndFieldName y email)))) (sendEmail)))
```

clara with the subject “requested information” and the body containing charlie’s email address.

In the first phase of the algorithm (rows 1-6), the algorithm will find the following spans: “charlie’s email” which can be evaluated as “(getFieldByInstanceNameAndFieldName charlie email)”, with a span from 2 to 5, excluding 5 (the tokenizer returns ’s as a separate token), and a semantic category of “Field”; charlie, which is a contact and can therefore be evaluated simply as “charlie”, with a span from 2 to 3, and a semantic category of “InstanceName”; and “clara”, which can be evaluated as “clara”, with a span from 9 to 10, and a semantic category of “InstanceName”. We note that all three expressions are a subexpression of the logical form provided above (row 5). All these expressions will be added to C (row 6). Since “charlie” is completely contained by “charlie’s email”, it is removed from the set of candidates (C) (row 7).

In the second phase of the algorithm (rows 8-18), the algorithm will consider the following subsets of C, {{“charelie’s email”}, {“clara”}, {“charlie’s email”, “clara”}}. We note that the spans do not overlap. Since both {“clara”} and {“charlie’s email”} are each a strict subset of {“charlie’s email”, “clara”}, they are both discarded, and the algorithm will only consider {“charlie’s email”, “clara”} (row 9). The algorithm will then replace “(getFieldByInstanceNameAndFieldName charlie email)” in the provided logical form by a parameter (x), and in the second iteration it will replace also “clara” in the logical form by a second parameter (y) (rows 11-15). The logical form will then become: “(lambda x y (doSeq (doSeq (doSeq (doSeq (createInstanceEmail email) (setFieldFromString (getMutableFieldByFieldName subject) (stringValue "requested information"))) (setFieldFromFieldVal (getMutableFieldByFieldName body) (evalField x))) (setFieldFromFieldVal (getMutableFieldByFieldName recipient_list) (evalField (getFieldByInstanceNameAndFieldName y email)))) (sendEmail)))". Figure 4 illustrates the substitutions performed by the algorithm. After finding all the arguments, the algorithm searches for the activation word, i.e. the word that will be associated with this command in the lexicon and will activate the command (rows 17-18). This activation word must not be a word that is already used as part of an argument (e.g. "clara") nor can it be a stopword. Therefore, the algorithm finds the following activation words: "obtain" and "transmit". Each of these activation words are added to the lexicon with the logical form learned above. The semantic category is induced by the semantic categories of the arguments and the order of them in the sentence in relation to the activation word.

Eventually, the learned command can be called with any other argument, as long as it matches the type. All of the following examples will work (assuming all fields are defined and the values are present): "obtain tammy's address and transmit to tom", "obtain tammy's address to tom", "chocolate's ingredients transmit to bob".

Algorithm 1 Lexicon induction for a taught command.

```
Input: $s$ - A command containing $n$ tokens.
    $\ell$ - The user-provided logical form for the command.
    $b$ - Beam size.
Output: $\Lambda$ - A set of induced lexicon entries.
    Phase 1: Find spans of $s$ that can be parsed to logical forms that are subexpressions of $\ell$.
 1: Parse $s$ with beam search to produce $b$ logical forms $\lambda_{i,j,k}$ for every span $(i, j)$, $(k \in [1, ..., b])$ .
 2: $C \leftarrow \{\}$ // Set of candidate spans
 3: for each span $(i, j) : 0 \leq i < j \leq n$ do
 4:     for $k : 1 < k \leq b$ do
 5:         if isSubexpression($\lambda_{i,j,k}, \ell$) then
 6:             $C \leftarrow C \cup \{(i, j, k)\}$
 7: Discard elements of $C$ whose sentence span is completely contained by a larger candidate.
    Phase 2: Generate lexicon entries by extracting arguments in $C$ from $\ell$.
 8: $\Lambda \leftarrow \{\}$
 9: for each maximal subset $S$ of $C$ with non-overlapping spans do
10:     $\ell' \leftarrow \ell$
11:     for each $(i, j, k) \in S$ do
12:         // Replace $\lambda_{i,j,k}$ in $\ell'$ with an argument variable $a$.
13:         $a \leftarrow$ getUniqueVariableName()
14:         body $\leftarrow$ substitute($\lambda_{i,j,k}, \ell', a$)
15:         $\ell' \leftarrow$ (lambda $a$ body)
16:     for each $m : 0 \leq m < n$ do
17:         if $m$ is not contained by a span in $S$ and $s_m$ is not a stopword then
18:             $\Lambda \leftarrow \Lambda \cup \{s_m := \ell'\}$
```

The use of (unknownCommand) to begin the teaching interaction allows the user to provide all commands naturally, as if LIA can execute any command. The teaching interaction is triggered only when the user enters language the parser does not understand. It also ensures that taught commands are provided the same way that the user wants to use them. LIA also allows the user to explicitly initiate a teaching interaction by using a command such as "teach a command," which is parsed to (teachNewCommand). In this case, LIA responds "I'm happy to hear that you want to teach me a new command. Now say the command the way you would use it , then I will ask you what exactly to do in that case. I will try to generalize to similar sentences."

4 Evaluation

In order to evaluate the performance of LIA we ran a controlled experiment with human users. We evaluated LIA's performance with respect to several aspects and attempted to answer the following evaluation questions:

- Will human users manage to interact with LIA and teach it new commands without explicitly being asked to do so (voluntarily)?
- Will users interacting with LIA outperform users from the control group (in terms of completion time)?
- Will users interacting with LIA believe the interaction to be more interesting than those interacting with the non-learning agent? Will they perceive LIA's capabilities as novel and would they like an agent with similar capabilities on a smart device?

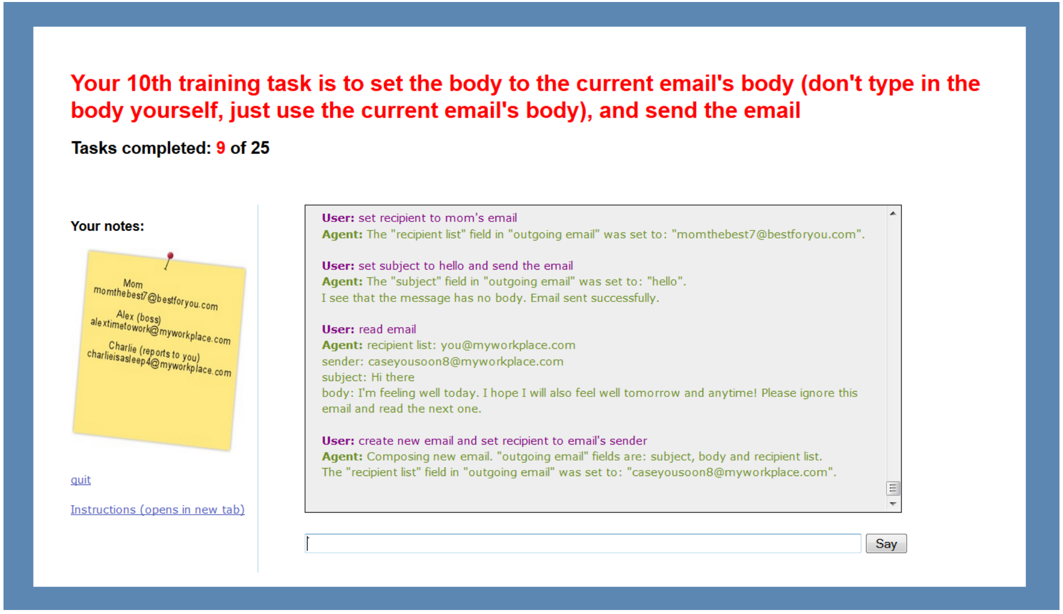
4.1 Experimental Setup

We conducted a user study with 131 users from Amazon Mechanical Turk to evaluate our agent. The set of users consisted of 63 (48.1%) females and 68 (51.9%) males. Subjects' ages ranged from 20 to 71, with a mean of 36.8. All users were residents of the USA. The users were randomly assigned to one of the following treatment groups:

1. *Learning Agent*: the users assigned to this group interacted with LIA. When users in this group gave the agent a command that it did not know how to execute, they were asked whether they would like to teach the agent this command.
2. *Non-Learning Agent*: the users assigned to this group interacted with a modified version of LIA that could not be taught new commands. Users in this group that gave the agent a command that it did not know how to execute, were told that the agent does not understand the command and were not offered to teach the agent how to perform this command.

The users had to fill out a short demographic questionnaire and sign the consent form. The users received instructions for the experiment and were required to answer a short quiz to ensure that they read the instructions. The quiz included between 5 to 6 (easy) multiple choice questions. Only after answering all the questions correctly they participated in the experiment. Users who made any mistake on one (or more) of the questions were asked to go back and correct their mistake. The questions were intended to ensure that the users understood that they were interacting in a mock environment (and not sending real emails), that they understood that they will be communicating with an agent, by text, and that they need to avoid spelling mistakes and type email addresses correctly. One question that was related to learning new commands was only shown to the users assigned to the 'learning agent' group. In this question the users had to select the option that stated that the agent may learn new commands, but that they are not required to teach it new commands. This question had two additional options that were considered incorrect if picked by the user; the first stated that the agent cannot learn new commands at all, and the second stated that they were required to teach the agent many new commands in order to complete the task.

Fig. 5 A screen-shot of a user interacting with LIA, using the Mechanical Turk web interface. The training task is shown on top (in red) and communication with LIA is performed using the text pane on the bottom. The previous commands appear above (in purple) along with LIA's responses (in green). The user's notes (email addresses required for the experiment) appear on the left.



Your 10th training task is to set the body to the current email's body (don't type in the body yourself, just use the current email's body), and send the email

Tasks completed: 9 of 25

Your notes:

Mom
momthebest7@bestforyou.com
Alex (boss)
alextimetowork@myworkplace.com
Charlie (reports to you)
charlieisasleep4@myworkplace.com

User: set recipient to mom's email
Agent: The "recipient list" field in "outgoing email" was set to: "momthebest7@bestforyou.com".

User: set subject to hello and send the email
Agent: The "subject" field in "outgoing email" was set to: "hello".
I see that the message has no body. Email sent successfully.

User: read email
Agent: recipient list: you@myworkplace.com
sender: caseyousoon8@myworkplace.com
subject: Hi there
body: I'm feeling well today. I hope I will also feel well tomorrow and anytime! Please ignore this email and read the next one.

User: create new email and set recipient to email's sender
Agent: Composing new email. "outgoing email" fields are: subject, body and recipient list.
The "recipient list" field in "outgoing email" was set to: "caseyousoon8@myworkplace.com".

No.	Task
1	Have the agent define/create the concept "contact" (simply say: define the concept contact)
2	Have the agent add the "email" field to the concept "contact"
3	**Create** a contact for mom (simply tell the agent to create a contact for mom)
4	**Set** mom's email correctly. (Set mom's email to the email that appears in the "notes" image.)
5	Ask the agent for mom's email
6	Create a new outgoing email
7	Set the recipient to mom's email and set the subject to hello and send the email
8	Read the current email (in the inbox). Ignore the email's content for now
9	Create a new email and set the recipient to the current email's sender
10	Set the body to the current email's body and send the email
11	Move to the **next** email (in the inbox). (No need to read it now.)
12	Move to the **previous** email (in the inbox). (No need to read it now.)
13	Teach the agent a **new command**: having it both move on to the next email and read it

Table 4 Training tasks the users were required to complete before approaching the main task. Task (13) was only given to users in the 'learning agent' group.

The interaction page (see Figure 5) included a training phase consisting of 12 tasks that appeared on the top of the screen. Users in the 'learning agent' group were assigned an additional training task, in which they had to teach the agent a new command. The training tasks were designed in a way that anyone who completes all training tasks should be able to complete the main task. See Table 4 for a list of these tasks. After completing all training tasks, the users were given the main task. In the main tasks, the users were requested to read each incoming email and, for each email, follow the sender's request. There were 12 main

Sender	Email Content (*Subject* and Body)
Alex	*Task I asked* Are you working on the task that I asked you to work on? Please reply immediately.
Casey	*What to do?* I'm done with all my tasks, what should I do next? Please reply as soon as possible.
Charlie	*Tell Alex that I'm on my way* Please email Alex saying that I'm on my way.
Mom	*Family Event* You must ask your boss to approve your vacation for the family event on September 28th. Forward this email to your boss.
Alex	*Your Vacation* Your vacation has been approved. Please forward this email to your mom.

Table 5 Examples of emails that the users had to deal with during the main task.

tasks, corresponding to requests in 12 emails. Table 5 presents several examples of such emails. The email corpus included two additional emails that appeared at the end, for the case that a user skipped one of the emails, or failed to complete the request appearing in it. That is, users could skip up to two emails and still complete all the main tasks. The users were instructed to include the same subject when replying and forwarding an email, and include the same body when forwarding an email. A verification component was developed in order to determine whether a user accomplished a task. This component took into account the inner state of LIA and the actions that it had actually performed. Once a user completed a task, the user was notified and was given the next task.

None of the 12 main tasks *required* the user to teach LIA new commands; however, as we will show, users taught the system new commands that they believed would be useful for future tasks. Users could quit the experiment at any time using the "quit" button. Users who quit the experiment received full payment for participation, but did not receive an extra completion bonus. The appendix presents a sample of a user's interaction with LIA that includes teaching new commands.

After completing all tasks or clicking the "quit" button, the users received an ending questionnaire asking about their acquaintance with programming. The users could choose one of the following: "None", "Very little", "Some background from high-school", "Some background from college/university", "Bachelors (or other degree) with a major or minor in software, electrical or computer engineering or similar", and 'Significant knowledge, but mostly from other sources".

The users were also asked to assign their level of agreement with each of the following sentences on a 7-point Likert scale [29] (strongly disagree, disagree, slightly disagree, neither agree nor disagree, slightly agree, agree, strongly agree): (1) "The interaction with the agent was interesting"; (2) "The capabilities of the agent seem novel"; and (3) "I would like an agent with similar capabilities on a smart device".

4.2 Results

We compare the performance of the users in the 'learning agent' group with those in the 'non-learning agent' group on the completion rate of all the tasks (see Table 6), the completion time of the tasks (see Table 7), and the users' ratings that were obtained from the Likert scale questionnaire (see Table 8). We use a single tail student's t-test to compare the

User Group	Number of users	Completion Rate
‘Learning agent’ group	53	47.1%
‘Non-learning agent’ group	78	33.3%

Table 6 Completion rate (percent of users who completed all 12 main tasks).

Criterion	Completion Time Average (STD)	
	‘Learning Agent’ Group	‘Non-Learning Agent’ Group
Training Completion Time	8:58 (4:01)	7:44 (4:15)
Main Task Completion Time	**21:53*** (8:20)	34:29 (24:33)
Total Time	**30:51*** (11:26)	42:13 (26:32)

Table 7 A comparison of the completion time between the users assigned to the ‘learning agent’ group and those assigned to the ‘non-learning agent’ group. Standard deviation appear in parenthesis. Asterisk indicates statistical significance (with $p < 0.05$).

two groups, as we expected the users interacting with LIA to complete the task faster, and assign higher rankings to the Likert scale questions.

4.2.1 Completion Rates

The assignment to each of the treatment group was performed randomly per user, that is, each user was randomly assigned to one of the two groups regardless of previous assignments. Therefore, both treatment groups did not result in the same size. 53 users were randomly assigned to the ‘learning agent’ group, while 78 users were randomly assigned to the ‘non-learning agent’ group.

The tasks that the users were required to complete in this experiment were very different from ordinary Mechanical Turks tasks. Completing all 12 main tasks (in addition to completing all training tasks) was not trivial and required substantial effort on the part of the users. The verification mechanism required the users to complete each task before they could move on to the next one. Therefore, it was expected that not all users would perform as well and complete all the tasks.

25 users (47.1%) completed all tasks successfully in the ‘learning agent’ group, and 26 users (33.3%) completed all tasks in the ‘non-learning agent’ group. While these numbers may seem low, we would like to note that it is well known that the primary goal of Mechanical Turk users is income generation [27,15]. Therefore, these numbers indicate that many users believed that the time it would take them to complete all the tasks would not be worth the bonus amount, and therefore they clicked on the quit button in order to obtain the participation payment. The differences between the two groups are not statistically significant. However, as we later show, users interacting with LIA rated the task as more interesting than those interacting with the non learning agent, and completed the task faster; this might explain why more users in the group that interacted with LIA decided to complete all tasks and not quit in the middle.

4.2.2 Completion Time

Table 7 presents the average completion time for each of the treatment groups (considering only users that completed all tasks). Users assigned to the ‘learning agent’ group spent significantly less time ($p = 0.0084 < 0.05$) on the main task than those in the ‘non-learning

Evaluation Criterion	Mean (STD) of User Ratings (1-7)			
	Learning Agent		Non-Learning Agent	
	Completers	All Users	Completers	All Users
The interaction with the agent was interesting	**5.3*** (1.3)	4.83* (1.6)	4.52 (1.8)	4.04 (2.0)
The capabilities of the agent seem novel	**5.0** (1.2)	4.02 (1.6)	4.41 (1.7)	3.84 (1.9)
I would like an agent with similar capabilities on a smart device	**4.83** (1.4)	4.08 (1.6)	4.16 (1.6)	4.08 (1.8)

Table 8 Users' rating on Likert scale questions testing users' attitude towards the agent (obtained from final questionnaire). This table compares the ratings of users who interacted with LIA and those who interacted with a non-learning version of LIA. The 'Completers' column indicates the rating provided by users who completed all tasks and did not quit in the middle. Standard deviation appear in parenthesis. Asterisk indicates statistical significance (with $p < 0.05$) when comparing the 'learning agent' group with the 'non-learning agent' group.

agent' group (using the student's one tail t-test). While the users in the 'non-learning' agent spent less time on the training task, this is not surprising, as they had one less task to complete (they were not required to teach the agent something). Despite spending a longer period of time on the training tasks, users assigned to the 'learning agent' group spent significantly less time ($p = 0.29 < 0.05$) on the whole task (including the training phase). This result indicates that teaching an agent new commands results in a significantly time saving, despite the time accounted for teaching the agent new commands.

4.2.3 User Ratings

We now turn to compare the user ratings on the Likert questions. Table 8 compares the average and standard deviation of the ratings provided by the users in the 'learning agent' group with the ratings provided by the users in the 'non-learning agent' group. The 'Completers' column indicates the rating provided by users who completed all tasks and did not quit in the middle. As shown in the table, the users in the 'learning agent' group thought that the interaction with the agent was significantly more interesting than those in the 'non-learning agent' group ($p < 0.05$). This is true when considering all the users ($p = 0.011$) and when considering only users who completed all tasks ($p = 0.047$). Users interacting with the learning agent who completed all tasks seem to rank the agent higher also on the remaining two questions; these results only indicate a statistical trend with ($p = 0.09 < 0.1$ and $p = 0.07 < 0.1$).

4.2.4 Commands Taught

Of the 25 users in the learning agent group who completed all tasks, 22 users (88%) taught LIA new commands during the main tasks. The users interacting with the learning agent taught, on average 2.52 new commands (during the main tasks), with a standard deviation of 1.23. New commands were used on average 3.7 additional times (with a large standard deviation of 4.38). Users interacting with the learning agent reused taught commands 9.6 times on average (per user), with a standard deviation of 7.67.

5 Dataset

Experiments conducted with LIA resulted in a large dataset. We believe that such a dataset can serve as a good source for developing and testing natural language parsers. In this section

Table 9 Corpus statistics for the Email Assistant dataset

Statistical Criterion	Value
Number of user utterances	4759
User sessions	113
Avg length of session (utterances)	42
Word types (all utterances)	704

we describe our method for annotating a subset of this dataset and evaluate the performance of several baselines on it, including a contextual deep learning based parser.

5.1 Dataset Annotation

The data consists of sequences of user utterances and system responses. In order to make the data usable for research, we annotated user utterances with their associated logical forms. e.g., the utterance: *'What is Mom's email?'* is annotated with the logical form (evalField (getFieldByInstanceName mom email)), following the logical language from above. Utterances that could not be reasonably expected to be interpreted by the email agent were marked as unknownCommand. However, if the user later taught the system what she meant, future instances of the utterance were marked with the intended logical form (e.g., users often taught the command *'Next'* to read and move to the next email in the inbox). Sequences devolving into non-meaningful interactions were removed, e.g., if the annotator deemed that the user did not intend to complete a task. Superfluous segments of the original conversation (e.g., utterances re-phrasing previous utterances that the system did not process) were also manually pruned. Since LIA allows users to teach new procedures (e.g., forwarding an email), concepts (e.g., the concept of a contact, which has fields such as an email address, a phone number, etc.), and instances (e.g., instantiating a contact) on-the-fly, linguistic usage in the experiments is complex and diverse, compared to many existing datasets.

Annotating every command by manually specifying its logical form would require experience with the underlying logical language of the system. Instead, we developed a program that allowed faster annotation using an alternate procedure. The software allows annotators to load a conversation sequence, and execute each utterance with a live version of the email agent. After executing each utterance with the agent, the annotator obtains the agent's response. If this response indicates that the agent has correctly interpreted the command (judged by the annotator's belief of the user's intent), the annotator may save the associated logical form for the utterance, as parsed by the online agent. However, if the agent's response is not the response expected by the annotator, (indicating that the agent did not interpret the command correctly), the annotator may provide a command which (i) reflects the intention of the utterance, and that (ii) the agent can interpret correctly. In effect, this strategy uses annotators to paraphrase the original command into different natural language commands that the agent would parse to the correct logical form, without exposing them to the underlying meaning representation formalism. The annotation process took approximately 50 hours of work. Figure 9 summarizes the statistics for the curated dataset.

Table 10 Parser Accuracy

Parser Type	Accuracy
Lookup	0.552
Smart-Lookup	0.569
SPCon	0.623
WL-LSTM	0.702
WL-LSTM-WC	0.741
CL-LSTM	0.735
CL-LSTM-WC	**0.753**

5.2 Baseline Parsers

For evaluation and future comparisons, we split the data into a training fold (93 conversation sequences, 4005 utterances) and a test fold (20 conversation sequences, 754 utterances). We note that the original CCG-based method presented in this paper resulted in an accuracy of 0.727 on the test set. In addition, we evaluated the performance of the following 7 parsers on the data. These baselines are based on classical supervised learning and therefore cannot be directly compared to the original CCG-based method presented in this paper for several reasons. First, the CCG-based parser relies on manually crafted lexicons, second, it is based on a very small dataset (since the large dataset did not exist when running the experiment), and, most importantly, it was designed to learn new commands based on a single user example.

Lookup searches the training set for an exact match for the given utterance and if it finds such a match it returns the corresponding logical form. *Smart-Lookup* is identical to the lookup method, but if it finds more than a single exact match it returns the most common corresponding logical form and rather than failing when not finding an exact match, it outputs the most common output (unknownCommand in our case). *SPCon* is a method we developed in [56] which formulates semantic parsing of conversations as a structured prediction task, incorporating structural features that model the ‘flow of discourse’ across sequences of utterances. The remaining 4 parsers use a deep neural network based on a machine translation model such as the one described in [10] to directly map utterances to logical forms. That is, the user utterances are regarded as the source language, while the logical forms are considered to be the target language. The architecture of these methods is composed of 3 levels of LSTM cells with a size of 1024 each. We considered two different LSTM based models. The first was a word level (WL) model, in which our model created a dictionary of all the *words* in both the source and target, and then predicted the next word given the utterance and the previous words in the logical form that it has already predicted. The second model was a character level (CL) model. This model split the user utterance into characters. In this model the label space (logical forms), was split according to “program tokens”, that is function calls and parenthesis were considered a single token, but other text such as email addresses etc. was split into characters. Note that the LSTM parser was not given any additional domain knowledge, therefore, even parenthesis matching and matching the number of arguments to every function was not pre-programmed and had to be learned by the LSTM parser. With an attempt to further improve the parser accuracy we also tested the performance of the LSTM parser when provided with context (WC), that is, instead of the utterances containing only the current sentence said by the user, they also contained the previous two sentences that the user said.

Table 10 provides a comparison of the performance of the different methods. As depicted in the table, the LSTM parsers outperformed all other methods, with the character level parsers outperforming the word-level ones, and with the parsers that take the context into account outperforming those that do not.

6 Discussion and Future Work

LIA, the system presented in this paper, includes support only for the email domain. While LIA can be extended by instruction in natural language, this extension can only be based on LIA's 45 primitive function. That is, the user may only define new commands which are aliases or compositions of commands that are already a part of LIA. For example, a user can teach LIA a new command such as "write hello in subject" using the command "set subject to hello", or "notify Bob that the computer is broken" using the relevant commands without requiring a programmer, but the user cannot teach LIA a command such as "navigate home", as LIA currently has no notion of navigation, nor can it be composed out of its 45 primitive functions. This limitation is not much different than the development of any new software or hardware component, which requires an API for access. Therefore, in order to allow LIA to support additional sensors and effectors (e.g. calendar, GPS, etc.) a programmer would be required to allow access to these sensors and effector from the software. In addition, at least a single lexicon entry must be added to LIA's lexicon which will associate some command with the new sensor or effector. The users can then teach LIA new commands or aliases by using the command that is already associated with the new sensor.

There is, however, a short-cut which may allow users to access new sensors and effectors without the necessity of a programmer in the loop. This is by creating low-level primitive commands. For example, if LIA is used on a smartphone or a home assistant with a graphical user interface, these primitive functions may include clicking on a button, moving the cursor to some point in the screen, scrolling and typing something in an input box. The user can then teach higher level commands (e.g. "Order an Uber for 4pm") using these low level commands (e.g. "Click on the Uber app, click on the set a time option, enter 4pm and click on request now"). This solution does come at a cost though, as it may become more difficult to generalize between different users, apps (or app versions), and different smartphones or operating systems.

There are significant opportunities for future work. The most obvious direction, as mentioned above, is supporting more sensors and effectors, both physical and cyber, such as, calendar, caller ID, text messages, GPS, social networks and ride-sharing applications. When adding additional sensors and supporting speech input, LIA will confront the problem of handling additional ambiguity, which may require taking the context into account. For example, "tell Bob I'm working on it" may be translated to either send an email to Bob or send a text message to Bob; however, if the previous command was "read my email", the correct translation is likely to be send an email to Bob.

Deployment of the instructable agent to a community of users, may allow users to collaboratively define new commands. This raises questions such as generalizability, and stability. LIA and/or its users will need a method to evaluate new commands taught, and decide which should be elevated to the collective knowledge of the agent used by all users and which should remain with the specific user who taught it (since some commands may be personal, confusing, non intuitive and perhaps even adversary). Another direction for future work is to develop a method which will identify whether a command has actually completed what the user intended or not (this can be done by using similarity to consecutive commands, or attempts to undo). Using this data, the agent can improve its accuracy, both by updating its feature weights and by learning new lexicon entries.

In this current work, we focused solely on procedural execution, i.e. commands that translate to a list of other known commands which are serially executed (with the support of arguments and variables). In future work we intend to provide support also to more complex instructions including if-then clauses, which may be interpreted as rules (e.g. "if I receive an email during a meeting, and it is related to the meeting, then notify me immediately").

7 Conclusions

In this paper we present our Learning by Instruction Agent (LIA), which, by using CCG parsing, lambda calculus, and our novel lexicon induction method, is able to learn by instruction given in natural language in the email domain. LIA receives step by step natural language instructions from users, on how to implement a command (such as forwarding an email to a specific contact), and is able to generalize and later execute this command using different parameters (e.g. to forward a different email to a different contact). We show that with very little training (approximately 9 minutes), and with little or no programming knowledge, many users were able to interact with LIA and teach it new commands. We further show that users that interacted with LIA required, on average, significantly less time to complete all tasks than those interacting with a degenerated version of LIA that did not support learning new commands. We introduce an annotated dataset based upon interactions with LIA, and, using this dataset we present a deep learning base parser that achieves a good level of accuracy.

We suggest that although our current system is only a partial, initial implementation of an instructable agent, our approach can serve as a template for more elaborate systems. If instructable agents of this form could be made widely available to all users of mobile devices, the collective set of commands and natural language phrases that could be learned by instruction would quickly become quite extensive, changing the nature of mobile devices from systems that can perform only commands built into the system by their developers (or by 3rd party developers), into devices that can be instructed (programmed) by millions of users.

8 Acknowledgments

This work was supported in part by Samsung GRO, Verizon (Yahoo!) through CMU's In-Mind project [7] and DARPA under contract number FA8750-13-2-0005.

References

1. P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.
2. J. Allen, N. Chambers, G. Ferguson, L. Galescu, H. Jung, M. Swift, and W. Taysom. Plow: A collaborative task learning agent. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 1514. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.
3. O. Amir and Y. K. Gal. Plan recognition and visualization in exploratory learning environments. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 3(3):16, 2013.
4. B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

5. Y. Artzi, K. Lee, and L. Zettlemoyer. Broad-coverage ccg semantic parsing with AMR. In *EMNLP*, pages 1699–1710, 2015.
6. A. Azaria, Y. Gal, S. Kraus, and C. V. Goldman. Strategic advice provision in repeated human-agent interactions. *Autonomous Agents and Multi-Agent Systems*, 30(1):4–29, 2016.
7. A. Azaria and J. Hong. Recommender system with personality. In *RecSys*, pages 207–210, 2016.
8. A. Azaria, J. Krishnamurthy, and T. M. Mitchell. Instructable intelligent personal agent. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.
9. A. Azaria, A. Richardson, and S. Kraus. An agent for deception detection in discussion based environments. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, pages 218–227. ACM, 2015.
10. D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.
11. J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, volume 2, page 6, 2013.
12. A. W. Biermann. *Natural language programming*. Springer, 1983.
13. A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Robot programming by demonstration. In *Springer handbook of robotics*, pages 1371–1394. Springer, 2008.
14. S. R. Branavan, H. Chen, L. S. Zettlemoyer, and R. Barzilay. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 82–90. Association for Computational Linguistics, 2009.
15. R. Brewer, M. R. Morris, and A. M. Piper. Why would anybody do this?: Understanding older adults' motivations and challenges in crowd work. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 2246–2257. ACM, 2016.
16. C. Buntain, A. Azaria, and S. Kraus. Leveraging fee-based, imperfect advisors in human-agent games of trust. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.
17. S. Calinon, F. Guenter, and A. Billard. On learning, representing, and generalizing a task in a humanoid robot. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 37(2):286–298, 2007.
18. R. Cantrell, J. Benton, K. Talamadupula, S. Kambhampati, P. Schermerhorn, and M. Scheutz. Tell me when and why to do it! run-time planner model updates via natural language instruction. In *Human-Robot Interaction (HRI), 2012 7th ACM/IEEE International Conference on*, pages 471–478. IEEE, 2012.
19. M. Chalamish, D. Sarne, and S. Kraus. Mass programmed agents for simulating human strategies in large scale systems. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 135. ACM, 2007.
20. D. L. Chen and R. J. Mooney. Learning to sportscast: a test of grounded language acquisition. In *Proceedings of the 25th international conference on Machine learning*, pages 128–135. ACM, 2008.
21. D. L. Chen and R. J. Mooney. Learning to interpret natural language navigation instructions from observations. *San Francisco, CA*, pages 859–865, 2011.
22. M. Chkroun and A. Azaria. Lia: A virtual assistant that can be taught new commands by speech. *International Journal of Human–Computer Interaction*, pages 1–12, 2019.
23. W. J. Clancey. The epistemology of a rule-based expert systema framework for explanation. *Artificial intelligence*, 20(3):215–251, 1983.
24. J. Clarke, D. Goldwasser, M.-W. Chang, and D. Roth. Driving semantic parsing from the world's response. In *CoNLL*, pages 18–27. Association for Computational Linguistics, 2010.
25. G. Druck, G. Mann, and A. McCallum. Learning from labeled features using generalized expectation criteria. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 595–602. ACM, 2008.
26. N. Haas and G. G. Hendrix. An approach to acquiring and applying knowledge. In *AAAI*, pages 235–239, 1980.
27. K. Hara, A. Adams, K. Milland, S. Savage, C. Callison-Burch, and J. P. Bigham. A data-driven analysis of workers' earnings on amazon mechanical turk. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 449. ACM, 2018.
28. S. Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1):335–346, 1990.
29. T. R. Hinkin. A brief tutorial on the development of measures for use in survey questionnaires. *Organizational research methods*, 1(1):104–121, 1998.
30. W. B. Knox and P. Stone. Interactively shaping agents via human reinforcement: The tamer framework. In *Proceedings of the fifth international conference on Knowledge capture*, pages 9–16. ACM, 2009.
31. K. R. Koedinger, V. Aleven, N. Heffernan, B. McLaren, and M. Hockenberry. Opening the door to non-programmers: Authoring intelligent tutor behavior by demonstration. In *Intelligent Tutoring Systems*, pages 162–174. Springer, 2004.
32. D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

33. J. Krishnamurthy and T. Kollar. Jointly learning to parse and perceive: Connecting natural language to the physical world. *Transactions of Association for Computational Linguistics*, 2013.
34. J. Krishnamurthy and T. M. Mitchell. Weakly supervised training of semantic parsers. In *EMNLP-CoNLL*, pages 754–765, 2012.
35. K. Kuklinski, K. Fischer, I. Marhenke, F. Kirstein, D. Solvason, N. Kruger, T. R. Savarimuthu, et al. Teleoperation for learning by demonstration: Data glove versus object manipulation for intuitive robot control. In *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2014 6th International Congress on*, pages 346–351. IEEE, 2014.
36. T. Kwiatkowski, E. Choi, Y. Artzi, and L. S. Zettlemoyer. Scaling semantic parsers with on-the-fly ontology matching. In *ACL*, 2013.
37. T. Kwiatkowski, L. Zettlemoyer, S. Goldwater, and M. Steedman. Inducing probabilistic ccg grammars from logical form with higher-order unification. In *EMNLP*, pages 1223–1233, 2010.
38. V. Le, S. Gulwani, and Z. Su. Smartsynth: Synthesizing smartphone automation scripts from natural language. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 193–206. ACM, 2013.
39. T. J.-J. Li, A. Azaria, and B. A. Myers. Sugilite: creating multimodal smartphone automation by demonstration. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 6038–6049. ACM, 2017.
40. C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014.
41. C. Matuszek, N. FitzGerald, L. Zettlemoyer, L. Bo, and D. Fox. A joint model of language and perception for grounded attribute learning. *arXiv preprint arXiv:1206.6423*, 2012.
42. B. A. Myers, A. Cypher, D. Maulsby, D. C. Smith, and B. Shneiderman. Demonstrational interfaces: Coming soon? In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 393–396. ACM, 1991.
43. S. Nakaoka, A. Nakazawa, F. Kanehiro, K. Kaneko, M. Morisawa, H. Hirukawa, and K. Ikeuchi. Learning from observation paradigm: Leg task models for enabling a biped humanoid robot to imitate human dances. *The International Journal of Robotics Research*, 26(8):829–844, 2007.
44. P. Pasupat and P. Liang. Compositional semantic parsing on semi-structured tables. *arXiv preprint arXiv:1508.00305*, 2015.
45. B. Price and C. Boutilier. Accelerating reinforcement learning through implicit imitation. *Journal of Artificial Intelligence Research*, 19:569–629, 2003.
46. C. Quirk, R. Mooney, and M. Galley. Language to code: Learning semantic parsers for if-this-then-that recipes. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL-15)*, pages 878–888, Beijing, China, July 2015.
47. C. Quirk, R. J. Mooney, and M. Galley. Language to code: Learning semantic parsers for if-this-then-that recipes. In *ACL (1)*, pages 878–888, 2015.
48. A. Reed. *Creating Interactive Fiction with Inform 7*. Cengage Learning, 2010.
49. A. Rosenfeld, M. Cohen, M. E. Taylor, and S. Kraus. Leveraging human knowledge in tabular reinforcement learning: A study of human subjects. *The Knowledge Engineering Review*, 33, 2018.
50. A. Rosenfeld and S. Kraus. Predicting human decision-making: From prediction to action. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 12(1):1–150, 2018.
51. A. Rosenfeld, I. Zuckerman, E. Segal-Halevi, O. Drein, and S. Kraus. Negochat-a: a chat-based negotiation agent with bounded rationality. *Autonomous Agents and Multi-Agent Systems*, 30(1):60–81, 2016.
52. E. Sarafian, A. Tamar, and S. Kraus. Safe policy learning from observations. In *The Multi-disciplinary Conference on Reinforcement Learning and Decision Making*, 2019.
53. A. Segal, Y. B. David, J. J. Williams, K. Gal, and Y. Shalom. Combining difficulty ranking with multi-armed bandits to sequence educational content. In *International Conference on Artificial Intelligence in Education*, pages 317–321. Springer, 2018.
54. L. She, S. Yang, Y. Cheng, Y. Jia, J. Chai, and N. Xi. Back to the blocks world: Learning new actions through situated human-robot dialogue. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, pages 89–97, 2014.
55. Y. Spielberg and A. Azaria. The concept of criticality in reinforcement learning. *International Conference on Tools with Artificial Intelligence*, 2019.
56. S. Srivastava, A. Azaria, and T. Mitchell. Parsing natural language conversations using contextual cues. In *IJCAI 2017*, 2017.
57. M. Steedman and J. Baldridge. Combinatory categorial grammar. *Non-Transformational Syntax: Formal and Explicit Models of Grammar*, pages 181–224, 2011.

58. J. Thomason, S. Zhang, R. Mooney, and P. Stone. Learning to interpret natural language commands through human-robot dialog. In *Proceedings of the Twenty-Fourth international joint conference on Artificial Intelligence (IJCAI)*, 2015.
59. Y. Wang, J. Berant, and P. Liang. Building a semantic parser overnight. In *ACL*, 2015.
60. T. Winograd. A procedural model of language understanding. 1973.
61. Y. W. Wong and R. J. Mooney. Learning synchronous grammars for semantic parsing with lambda calculus. In *ACL*, volume 45, page 960, 2007.
62. L. S. Zettlemoyer and M. Collins. Learning to map sentences to logical form: structured classification with probabilistic categorial grammars. In *UAI '05, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence*, 2005.
63. L. S. Zettlemoyer and M. Collins. Online learning of relaxed ccg grammars for parsing to logical form. In *EMNLP-CoNLL*, pages 678–687, 2007.

9 Appendix

The following is a sample of a real user interacting with LIA:

User: go
Agent: Sorry, but I don't understand. Would you like to teach me what to do in this case (either say yes or simply ignore this question)?
User: yes
Agent: Great! When you say, for example: "go", what shall I do first?
User: move to next email and read it
Agent: Set to next incoming email successfully. sender: caseyousoon8@myworkplace.com subject: Another email body: I felt like sending you another email. Please ignore this email too and read the next one. What shall I do next (when executing: "go")?
User: end
Agent: I'm currently learning the new command ("go"). I'm trying to generalize to other similar commands, this takes several seconds...
Agent: I now know what to do when you say (for example): "go"!
...
User: go
Agent: Set to next incoming email successfully. recipient list: you@myworkplace.com sender: caseyousoon8@myworkplace.com subject: Charlie body: I asked Charlie to do what you said, but I see that it must come from you. Please forward this email to Charlie.
User: create contact Charlie
Agent: Instance "charlie" (of concept "contact") was created. "contact" fields are: email.
User: set Charlie's email to charlieisasleep4@myworkplace.com
Agent: The "email" field in "charlie" was set to: "charlieisasleep4@myworkplace.com".
...
User:go
Agent: Set to next incoming email successfully. sender: alextimetowork@myworkplace.com subject: Work before parting body: We will all have to work very hard next Monday, Tuesday and Wednesday. Please forward this email to Charlie.
User: forward charlie
Agent: Sorry, but I don't understand. Would you like to teach me what to do in this case?
User: yes
Agent: Great! When you say, for example: "forward charlie", what shall I do first?
User: set recipient list to charlie's email, set subject to current email's subject, set body to current email's

Agent: I see that there is no email being composed. Do you want to compose a new email?
User: create new email, set recipient list to charlie's email, set subject to current email's subject, set body to current email's body, send email
Agent: Composing new email. The "recipient list" field in "outgoing email" was set to: "charlieisasleep4@myworkplace.com". The "subject" field in "outgoing email" was set to: "Work before parting". The "body" field in "outgoing email" was set to: "We will all have to work very hard next Monday, Tuesday and Wednesday. Please forward this email to Charlie.". Email sent successfully. What shall I do next (when executing: "forward charlie")?
User: end
Agent: I now know what to do when you say (for example): "forward charlie"!